Int J Digit Libr (2009) 10:159–177
DOI 10.1007/s00799-010-0058-0

Dependency management for digital preservation using semantic web technologies

Yannis Marketakis · Yannis Tzitzikas

Published online: 21 May 2010

Abstract The preservation of digital objects is a topic of prominent importance for archives and digital libraries. In this article, we focus on the problem of preserving the *intelligibility* of digital objects. We formalize the problem in terms of dependencies and specify a number of basic intelligibility-related tasks. In parallel, we introduce a preservation scenario as a means for clarifying the pros and cons of various representation and modeling languages that are used for the problem at hand, which reveals the benefits of adopting Semantic Web (SW) languages as a representation framework. To this end, we propose a minimal core ontology for representing intelligibility-related dependencies along with methodological hints for extending it. Finally, we report empirical and experimental results from applying the proposed approach on real data sets. It is worth mentioning that this approach can be used not only on SW-based repositories or archives, but also on those that are based on conventional approaches and languages (like EAST, DEDSL, XFDU/SAFE).

Keywords Digital Preservation · Intelligibility · Dependency management

1 Introduction

Modern society and economy is increasingly dependent on a deluge of only digitally available information. The preservation of digital information within an unstable and rapidly evolving technological (and social) environment is a challenging problem of prominent importance. In recent years, there has been a growing interest around this problem and several aspects of the problem are being investigated, including metadata and services for preservation (e.g., see [7,12]), data/medium preservation approaches (e.g., [11,5,21,23]), migration and encapsulation approaches (e.g., [11,30,13,9]), workflow and preservation approaches (e.g., [27]), theoretical attempts (e.g., see [4]), cost-related strategies for data preservation planning (e.g., see [17,36,28,29,32]), standards (like OAIS [14]), and there are several ongoing international projects (like CASPAR [1] and PLANETS [26]).

Given a corpus of digital information, the first rising question is: *what should we preserve and how?* Certainly, we have to preserve the bits of digital objects. However we should also try to preserve their (a) accessibility, (b) integrity, (c) authenticity, (d) provenance, and (e) intelligibility (by human or artificial actors). In this article, we elaborate on the aspect of *intelligibility*.

A formalization of the problem of preserving the intelligibility of digital objects that is based on dependencies was given in [37,38] and [39]. In this article, we extend that work and report our experiences from applying this model in practical (real world) cases. To the end, we discuss an extension of the model where dependencies are *goal*-oriented. Subsequently, we present a set of basic services designed according to that model. The proposed model and the corresponding services can aid several tasks of archivists (or digital curators), including:

- the decision of what metadata need to be captured and stored,
- the reduction of the metadata that have to be archived, or delivered (as a response to queries) to the users, and

Y. Marketakis · Y. Tzitzikas
Computer Science Department, University of Crete, Heraklion, Crete, Greece
e-mail: marketak@ics.forth.gr

Y. Marketakis · Y. Tzitzikas (✉)
Institute of Computer Science, FORTH-ICS, Heraklion, Crete, Greece
e-mail: tzitzik@ics.forth.gr

– the identification of the objects that are in danger in case a module (e.g., a software component or a format) is becoming obsolete (or has been vanished).

For instance, suppose that we want to preserve digital files containing temperature measurements from various places on earth. For instance, assume that we have a file named `datafile20080903_12PM.txt` with the following contents:

```
25.130  35.325  30.2
25.100  35.161  28.9
25.180  35.333  29.3
```

Each row contains the longitude, the latitude, and the measured temperature (in Celsius degrees), while the suffix of the filename (i.e., 20080903_12PM) reveals the date and time of the measurement. Some of the rising questions are: (a) what kind of metadata do we have to add and in what format, (b) how much of them do we really need to capture and record (is there any disciplined way to control it), (c) how much metadata should we pack together if we want to archive or deliver such files to end users? Can we answer the above questions objectively, or we have to take into account the *designated community*, i.e., the community for which we want to preserve these files?

In this article, we approach these questions through a dependency management approach. Subsequently we show how the model can be realized using Semantic Web (SW) technologies. In brief, the benefits of adopting SW technologies is that they provide standard and semantically interpretable exchange formats, and that the knowledge artifacts expressed in these languages lend themselves to reuse and extension. To make this clear, we use the temperature measurements preservation scenario and show how different approaches and languages can support it.

Regarding software architecture and efficiency, this article discusses and reports experimental results for two implementation choices, one proprietary file system-based (with limited expressive power), and one based on Semantic Web technologies.

In a nutshell, the contribution of this article lies in:

– providing an ontology for representing dependencies and community knowledge and in proposing how it can be extended to capture goal-oriented dependencies,
– specifying a set of basic dependency management services that can be exploited for supporting the preservation of intelligibility of digital objects
– comparing different languages and standards that have been proposed for digital preservation, and
– detailing implementation approaches including Semantic Web technologies.

The article is organized as follows: Section 2 describes the model and its extension. Section 3 describes intelligibility-related services. Section 4 discusses modeling using SW languages. Section 5 sketches a methodology for expressing and exploiting intelligibility-related dependencies. Section 6 describes a preservation scenario and the pros and cons of various languages and approaches. Subsequently, Sect. 7 describes the software architecture, the functionality of the `GapMgr` tool, and reports experimental results over real and synthetic data sets. Section 8 discusses related work. Sect. 9 discusses various extensions. And finally, Sect. 10 concludes this article and identifies issues that are worth for further research.

2 Formal model (for intelligibility preservation)

According to the OAIS reference model [14], metadata are distinguished to various broad categories. One very important (for preservation purposes) category of metadata is named *Representation Information* (RI) which aims at enabling the conversion of a collection of bits to something useful. In brief, the RI of a digital object should comprise information about the Structure, the Semantics and the needed Algorithms for interpreting and managing a digital object. It follows that intelligibility is closely related to RI. Figure 1 shows one corresponding part of the information model of OAIS.

2.1 Basic Model

Here, we provide a brief introduction to the model introduced in [37,39] and [38]. The proposed model is a small formal model comprising three basic notions: *module*, *dependency*, and *profile*.

We adopt a very general interpretation for the term *module*. It can be a software or hardware component (i.e., a txt file, a hard disk), or a knowledge model expressed either formally or informally (i.e., an rdf ontology, the greek language in general). Modules may require the availability of other modules to become intelligible. This is modeled by a binary relation denoted by $>$, where $t > t'$ means that t depends on t'. By analyzing the dependencies in this way, we get chains or trees of dependencies. To limit the potentially extremely large (or even endless) dependency graph, which is also hard to express explicitly, the notion of designated community profile (DC profile, or just *profile*) is introduced. A user u (where user could be a community of users or a software agent), may know (has available) a set of modules denoted by $T(u)$ where $T(u) \subseteq \mathcal{T}$ ($\mathcal{T}$ is the set of all modules). This is called designated community profile. Recall that according to OAIS reference model, [14]:

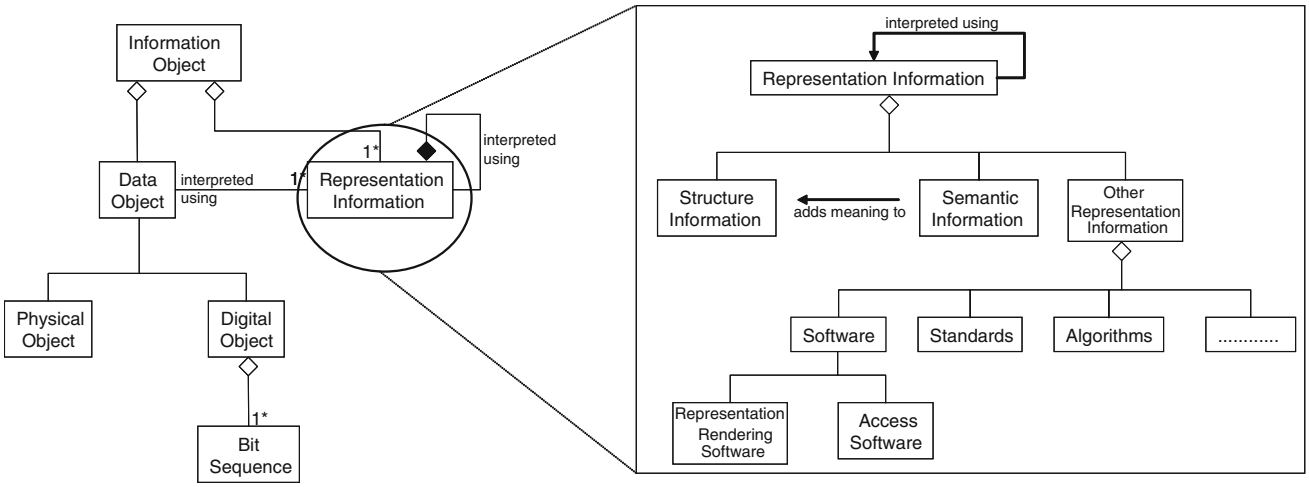

Fig. 1 The information model of OAIS

> A knowledge base is a set of information incorporated by a person or system that allows the person or system to understand received information

The above definition of knowledge base, relates to what we call *DC Profile* since it describes the knowledge assumed to be known. The key point is that DC profiles allow making these assumptions explicit by associating to each profile the modules which are assumed to be known to the users of that community. In other words, the quite abstract notion of "knowledge base" can be analyzed to more concrete elements each modeled at the desired (or feasible) level of detail.

The availability of dependencies allow us to define the *closure* of a module as follows:

Definition 1 The *closure* of a module t, denoted by $Nr^*(t)$, is t plus the set of all modules on which t depends directly or indirectly.

Some notations for various extensions of this definition (e.g., closures of sets of modules) that we use in the sequel are given in Table 1. Having defined the notion of closure, we can now proceed and state that a module t is *intelligible* by a user u if all the modules that are required by t belong to the closure of $T(u)$. If this does not hold, then t is not intelligible by u, and we have what we call *intelligibility gap*. Both *intelligibility* and *intelligibility gap* are defined rather straightforwardly.

However, we have to note at this point that we axiomatically assume that if a module t belongs to the profile of a user u, then it is assumed that all direct and indirect dependencies of t are also known by u.

Definition 2 A module t is *intelligible* by a user u if $Nr^+(t) \subseteq Nr^*(T(u))$. □

Table 1 Basic notions and notations

Notation	Definition	Name
$\mathcal{T}$	The set of all modules and objects	
t	An element of $\mathcal{T}$	A module
S	A subset of $\mathcal{T}$	A set of modules
$t > t'$	t Depends on t' (t requires t')	A dependency
$>^+$		The transitive closure ‡ of the binary relation $>$
$min_>(S)$	The minimal elements of S w.r.t. $>^+$	The minimals of S
$max_>(S)$	The maximal elements of S w.r.t. $>^+$	The maximals of S
$Nr(t)$	$= \{ t' \mid t > t' \}$	The direct dependencies of t, i.e., all modules on which t directly depends
$Br(t)$	$= \{ t' \mid t' > t \}$	The direct dependents on t, i.e., all modules that directly depend on t
$Nr^+(t)$	$= \{ t' \mid t >^+ t' \}$	All modules that t requires, i.e., the set of all direct and indirect dependencies of t
$Nr^*(t)$	$= \{t\} \cup Nr^+(t)$	The *closure of* t, i.e., t and all modules that t requires
$Nr^+(S)$	$= \cup_{t \in S} Nr^+(t)$	All modules required by S
$Nr^*(S)$	$= S \cup Nr^+(S) = \cup_{t \in S} Nr^*(t)$	The closure of the set S

‡ The transitive closure of a binary relation R is the smallest transitive relation that contains R

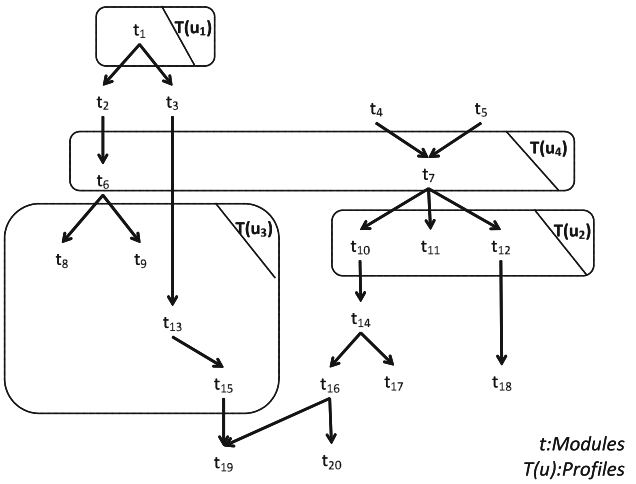

Fig. 2 Example of modules, dependencies and profiles

Definition 3 The *intelligibility gap* between a module t and the profile of a user u, is the minimum set of extra modules that u needs to have in order to understand t. This is denoted by $Gap(t, u)$, and is defined as:

$$Gap(t, u) = Nr^{+}(t) - Nr^{*}(T(u)). \quad \square$$

Here "−" denotes set difference. Clearly if t is *intelligible* by u then $Gap(t, u) = \emptyset$. The intelligibility gap is a useful notion for understanding what kind RI we need to have, to ensure the preservation of intelligibility of a digital object for a designated community. At the same time, it tells us what RI we need to deliver to a user in order to be sure that we give him an intelligible package (the package will contain the module and the required dependencies of this module). Figure 2 illustrates an example with $\mathcal{T} = \{t_1, \ldots, t_{20}\}$ and four profiles:

$$T(u_1) = \{t_1\}$$
$$T(u_2) = \{t_{10}, t_{11}, t_{12}, \}$$
$$T(u_3) = \{t_8, t_9, t_{13}, t_{15}\}$$
$$T(u_4) = \{t_6, t_7\}$$

Consider module t_1. Its direct dependencies are $Nr(t_1) = \{t_2, t_3\}$, while its closure is

$$Nr^{*}(t_1) = \{t_1, t_2, t_3, t_6, t_8, t_9, t_{13}, t_{15}, t_{19}\}.$$

Regarding intelligibility, notice that t_6 is intelligible by u_3 because $Nr^{+}(t_6) \subseteq Nr^{*}(T(u_3))$. However t_1 is not intelligible by u_3. Specifically, $Gap(t_1, u_3) = \{t_2, t_3, t_6\}$.

The availability of dependencies and community profiles allows deriving *packages*, either for archiving or for dissemination, that are *profile-aware*. For instance, OAIS [14] distinguishes packages to AIPs (Archival Information Packages) and DIPs (Dissemination Information Packages). The availability of explicitly stated dependencies and community profiles, enables the derivation of packages that contain exactly those dependencies that are needed so that the packages are intelligible by a particular DC profile.

Definition 4 The *(dissemination or archival) package* of a module t with respect to a user or community u, denoted by $Pack(t, u)$, is defined as:

$$Pack(t, u) = (t, Gap(t, u)) \quad \square$$

In our example,

$$Pack(t_1, u_1) = (t_1, \emptyset)$$
$$Pack(t_1, u_3) = (t_1, \{t_2, t_3, t_6\})$$
$$Pack(t_{15}, u_1) = (t_{15}, \emptyset)$$
$$Pack(t_{15}, u_2) = (t_{15}, \emptyset)$$

We have to note at this point that there is not any qualitative difference between DIPs and AIPs from our perspective. The only difference is that AIPs are formed with respect to the profile decided for the archive, which we can reasonably assume that it is usually more rich than user profiles. Details regarding the packaging formats are given in Sect. 6.

2.2 Addition of types

In this section, we motivate the need for introducing and supporting module and dependency *types*. Suppose that we want to preserve a file `a.java` containing java code. If our objective is to preserve the ability to read/edit this file, then its intelligibility depends only on the ASCII data format. If on the other hand, our objective is to be able to compile it (i.e., to preserve the compilability of the file), then it depends on the availability of a java compiler (i.e., `javac`), as well as on the availability of all other Java packages/classes that are used by `a.java` (i.e., all import statements). In general, we can say that:

It is the goal that determines which are the dependencies of a module.

For this reason, we introduce module and dependency types. Let C be the set of all module types, and D the set of all dependency types. If c is a subclass of c', we shall write $c \sqsubseteq c'$. If d is a subtype of a dependency type d', we shall write $d \sqsubseteq d'$. Clearly both $\sqsubseteq$ relations are transitive. Given a module t, we shall use $types(t)$ to denote the types (direct and indirect) of t. Given a direct dependency $t > t'$, we shall use $types(t > t')$ to denote the types (direct and indirect) of all direct dependencies between these two modules.

Having extended our model with types, we can now introduce *type-restricted dependency* gaps. As motivating example suppose that we have an object and we are interested only in its *compile* dependencies. In this case, and for computing the gap, we should traverse dependencies whose type is *compile*, or a subtype of *compile*. This can be defined formally as follows: given a module t, a profile u and a set of dependency types W ($W \subseteq D$), we define

$$t >_W t' \text{ iff (a) } t > t' \text{ and (b) } types(t > t') \cap W^* \neq \emptyset$$

where W^* is the set of all possible subtypes of W. The formula implies that we consider all types in W disjunctively. Now, we can define:

$$Gap(t, u, W) = \{ t' \mid t >_W^+ t' \} - Nr^*(T(u))$$

Since the number of dependency types may increase, it is beneficial to organize them according to an object-oriented approach, i.e., to introduce module types, and to specify the domain and range of each dependency type (each being a module type). For example, we can have a module type *Software* which is specialized to *SoftwareBinary* and *SoftwareSourceCode* types. Now the domain of the dependency type *compile* can be defined to be the class *SoftwareSourceCode*, while the range can be defined to be the module type *Software*. Apart from this, module types are useful due to the heterogeneity of modules in real settings. Finally, module types can also be exploited when computing closures or gaps, i.e., we could filter out modules based on type conditions.

3 Intelligibility (dependency) management services

In this section, we describe the main dependency management services based on the model described in Section 2. Specifically, we specify the basic query and update services.

3.1 Closures and gaps

Table 2 shows the definition of the services for computing gaps and closures and for deciding whether a module is intelligible by a profile or whether it depends (directly or indirectly) on another module.

3.2 Updating dependencies

The set of modules, dependencies, and profiles may change over time, and therefore we have to define a set of basic change operations. For this analysis, we first have to define *profile equivalence*.

Definition 5 Two profiles u and u' are equivalent, denoted by $u \sim u'$, if $Nr^*(T(u)) = Nr^*(T(u'))$. □

It follows that for every profile u it holds $T(u) \sim max_>(T(u))$, i.e., u is equivalent to a profile comprising only the maximal elements of $T(u)$ with respect to the dependency relation. So, we can reduce the size of a profile by keeping only its maximal elements. Hereafter, we assume that all profiles are stored in this way.

Table 3 introduces some basic services for *updating the dependency graph*. Each operation is specified by its pre- and post-condition. The post-conditions describe not only the conditions that must be true in the dependency graph, but also their side-effects on profiles. An archive could follow its own policy which may deviate from the one we describe (which is just indicative). The underlying assumption in our specification is that the dependencies are *transitive*. The list of operations includes an operation $upgradeModule(t)$ useful for creating a new module t_{new} which is a newer version of an existing module t. This operation is actually a shortcut which can be configured (or specialized according to the needs). For instance, if we assume that new versions are backwards compatible, then the new module t_{new} will have the same description with t (i.e., name, dependencies) but a revised version number. For example, if we issue $upgradeModule(t_{14})$ (assuming the example of Fig. 2), then we will get a new module t'_{14} which will also depend on t_{16} and t_{17}.

Since only the maximal elements of profiles are stored, the deletion of a dependency, say $delDependency(t, t')$, may "break" the defined profiles. To avoid such cases before executing $delDependency(t, t')$ we have to find those profiles that contain t or a broader term of t. If U is the set of all profiles, then the sought set of profiles, is $\{ u \in U \mid t \in Nr^*(T(u)) \}$. If u is a profile in the above set, we add to it the modules $Nr(t)$. This means that for the revised profile u' it holds that $T(u') = T(u) \cup Nr(t)$. After performing these updates, we can reduce the size of the profile to contain only

Table 2 Basic intelligibility services

Return value	Service	Definition
Module[]	Closure(Module[] S, DepTypes[] W)	$Nr^*_W = \cup_{t \in S}(\{t\} \cup \{ t' \mid t >_W^+ t' \})$
Module[]	Gap(Module t, Profile u, DepTypes[] W)	$Gap(t, u, W) = \{ t' \mid t >_W^+ t' \} - Nr^*(T(u))$
Boolean	isIntelligible(Module t, Profile u, DepTypes[] W)	$\{ t' \mid t >_W^+ t' \} \stackrel{?}{\subseteq} Nr^*(T(u))$

Table 3 Change operations

Ret. value	Operation	Pre-condition	Post-condition
Change operations on dependencies			
Boolean	addDependency (t, t′: Module, W: DepTypes[])	$t, t' \in \mathcal{T}, W \subseteq D$	For each $w \in W$ it holds: $t >_w t'$ and $>_w$ is acyclic
Boolean	delDependency (t,t′:Module, W: DepTypes[])	$t >_W t'$	For each $w \in W$ it holds: $t \not> _w t'$ For each $u \in U$, if $t \in Nr^*(T(u))$ then $T(u) \leftarrow max_>(T(u) \cup Nr(t))$
Boolean	delModule(t: Module)	$t \in T$	All dependencies that involve t are removed (by issuing delDependency operations) For each $u \in U$, $T(u) \leftarrow T(u) \backslash \{t\}$ $t \notin T$
Module	upgradeModule (t: Module)	$t \in T$	A new Module t_{new} is created such that: for each $w \in W$ if $t >_w t'$ then $t_{new} >_w t'$
Change operations on profiles			
Void	appendKnownModules (u:Profile, S: Module[])	$S \subseteq T$	$T(u) \leftarrow max_>(T(u) \cup S)$
Boolean	delModule (u:Profile, t: Module,)	$t \in Nr^*(T(u))$	$T(u) \leftarrow max_>((T(u) \cup Nr(t)) - \{t\})$

the maximal modules and delete the dependency $t > t'$. As an example, consider the profile u_3 of Fig. 2, where $T(u_3) = \{t_8, t_9, t_{13}, t_{15}\}$. Since $max_>(\{t_8, t_9, t_{13}, t_{15}\}) = \{t_8, t_9, t_{13}\}$, only these three modules need to be stored for that profile. Now suppose the deletion of the dependency $t_{13} > t_{15}$. To curate profile u_3, we need to add to it the modules in $Nr(t_{13})$, i.e., $\{t_{15}\}$.

3.3 Dependency management and ingestion quality control

The notions of profile and intelligibility gap allow reducing the amount of dependencies that have to be archived/delivered on the basis of DC profiles (recall how $Pack(t, u)$ is defined). Another aspect of the problem concerns the ingestion of information. Specifically, one rising question is whether we could provide a mechanism (during ingestion or curation) for identifying the representation information that is required or missing. This requirement can be tackled in several ways: (a) we require each module to be classified (directly or indirectly) to a particular class, so we define certain facets and require classification with respect to these (furthermore some of the attributes of these classes could be mandatory), (b) we define some dependency types as mandatory and provide notification services returning all those modules which do not have any dependency of that type, (c) we require that the dependencies of the objects should (directly or indirectly) point to one or several profiles. Below, we elaborate on policy (c).

Definition 6 A module t is *related* with a profile u, denoted by $t \mapsto u$, if $Nr^*(t) \cap Nr^*(T(u)) \neq \emptyset$.

This means that the direct/indirect dependencies of a module t lead to one or more elements of the profile u. At the application level (for more see Sect. 7.1), for each module t we can show all related and unrelated profiles, i.e.: $RelProf(t) = \{ u \in U \mid t \mapsto u\}$ and $UnRelProf(t) = \{ u \in U \mid t \not\mapsto u\}$ respectively. Note that $Gap(t, u)$ is empty if either t does not have any recorded dependency, or if t has dependencies but they are known by the profile u. The computation of the related profiles allows curators to distinguish these two cases ($RelProf(t) = \emptyset$ in the first and $RelProf(t) \neq \emptyset$ in the second). If $u \in RelProf(t)$ then this is just an indication that t has been described with respect to profile u, but it does not guarantee that its description is complete with respect to that profile.

4 Modeling using Semantic Web Languages

This section describes a realization of the proposed model and services using Semantic Web technologies.

4.1 Core ontology for dependencies

To represent modules, dependencies and DC profiles we defined an ontology (expressed in RDF/S [3]). Figure 3

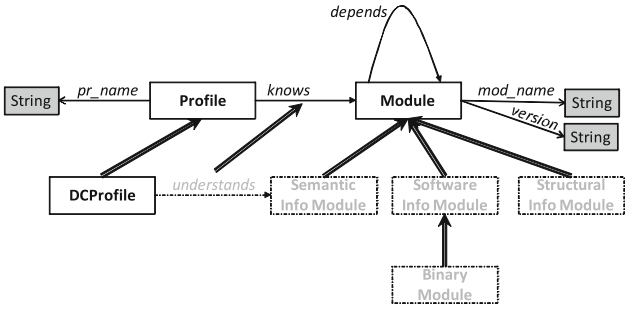

Fig. 3 The core ontology for dependencies

sketches the backbone of this ontology. We shall hereafter refer to this ontology with the name COD (Core Ontology for representing Dependencies). This ontology contains only the notion of `Profile` and `Module` and consists of only two RDF Classes and five RDF Properties (it does not define any module or dependency type). It can be used as a standard format for representing and exchanging information regarding modules, dependencies and DC profiles. Moreover it can guide the specification of the message types between the software components of a preservation information system.

Subsequently, we defined an ontology for module and dependency types. Currently, this ontology consists of 43 classes and 9 properties.[1] The typology of modules, as visualized by `StarLion`[2][40], is given in Fig. 4. This ontology extends the COD ontology.

One benefit of adopting Semantic Web languages is that we can base on COD the core query services of a preservation system. For instance, the queries that will be formulated using elements of COD, will continue to function correctly even if the data layer instantiates specializations of COD. Such specializations are described in the next section (Sect. 4.2), while Sect. 4.3 discusses how this ontology can be combined with other ontologies (which are not necessarily specializations of COD).

4.2 Extending the core ontology

Here, we motivate the need for specializing the core ontology. Recall the example with the file `a.java` containing java code. Figure 5 shows different kinds of dependencies for this example. Since the set of possible goals cannot be fixed in advance, we need a way to extend the set of goals and the corresponding dependency types. To this end, we propose representing goals by specializing the dependency relation (by exploiting "subPropertyOf" of RDFS). An example is shown in the left part of Fig. 5. Here, we define three new dependency types by simply specializing the one defined in the COD ontology. Specifically, a `_run` dependency is used if a module depends on another in order to be runnable, while the other two dependency types concern *compilability* and *editability*.

Notice that this approach is more extensible than other works (discussed in more detail in Sect. 8) that presuppose a fixed set of goals.

4.3 Combining COD with other ontologies

One rising question is how COD could be related with other ontologies that may have narrower, wider, overlapping, or orthogonal scope.

For instance, one may find out that an intelligibility dependency corresponds to a certain relationship, or a path of relationships, over another conceptual model (or ontology). Below, we discuss in brief such a case, assuming the CIDOC Conceptual Reference Model (ISO 21127) [18]. CIDOC CRM is a core ontology describing the underlying semantics of data schemata and structures from all museum disciplines and archives. It is the result of long-term interdisciplinary work and agreement and it has been derived by integrating (in a bottom-up manner) hundreds of metadata schemas. Consider a set of digital objects and suppose that we want to express their dependencies according to COD and their provenance (e.g., change of custody, or derivation history) according to CIDOC CRM. For instance, if one wants every CIDOC CRM `E73 Information Object` to be considered as `Module`, then he could "merge" these ontologies. Specifically he would define `E73 Information Object` as a subclass of `Module`. The ability of multiple-classification and of inheritance (of Semantic Web languages) gives this flexibility. Alternatively, one could either manually or through a declarative update language (e.g., RUL [20]), classify his data also with respect to COD. In that case COD could be considered as the schema of a read-only view of knowledge bases structured according to more sophisticated conceptual models. Just indicatively, Fig. 6 shows two approaches for modeling provenance. The first (a) is a naive approach where COD is just extended with the property *derivedBy*. The resulting structuring is very poor (for the needs of provenance). The second (b) shows a combination of COD with CIDOC CRM Digital Ontology[3] which is an extension appropriate for capturing the properties and the provenance of digital objects [35]. Both modules are classified to the class `E73 Information Object`. The provenance of the module `a.class` has been modeled using elements (classes and properties) of CIDOC CRM Digital, specifically using the class *Formal Derivation* and its properties (for more refer to [35]).

[1] It is available from http://www.casparpreserves.eu/.

[2] www.ics.forth.gr/~tzitzik/starlion.

[3] http://cidoc.ics.forth.gr/rdfs/caspar/cidoc_digital2.3.rdfs.

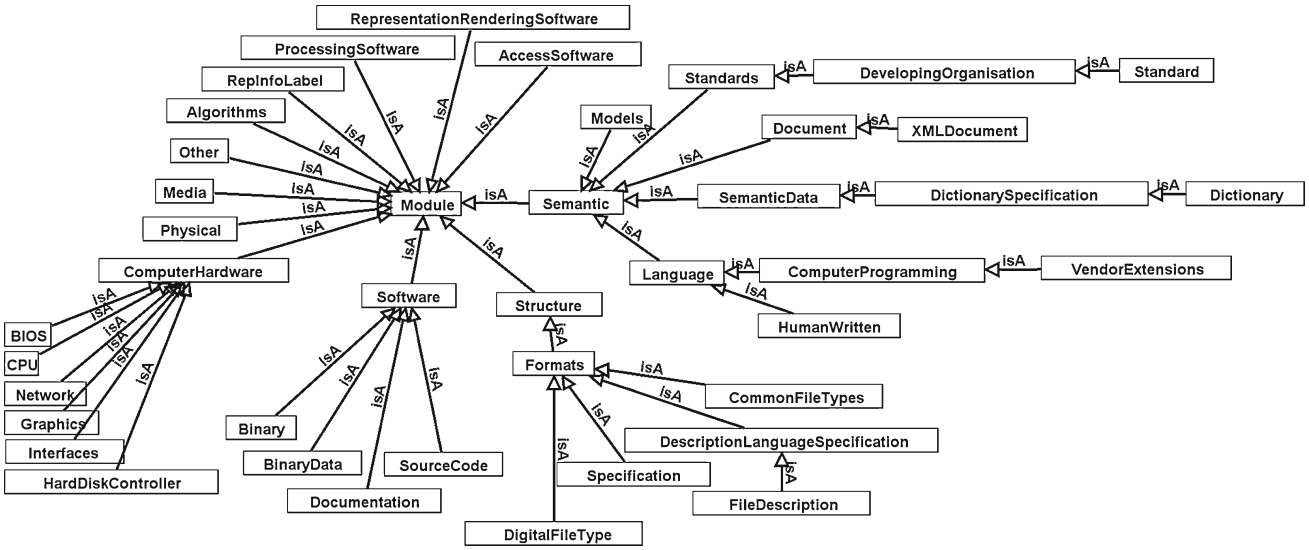

Fig. 4 Typology of modules

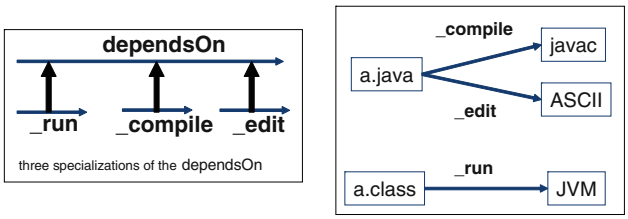

Fig. 5 Specializing dependency types

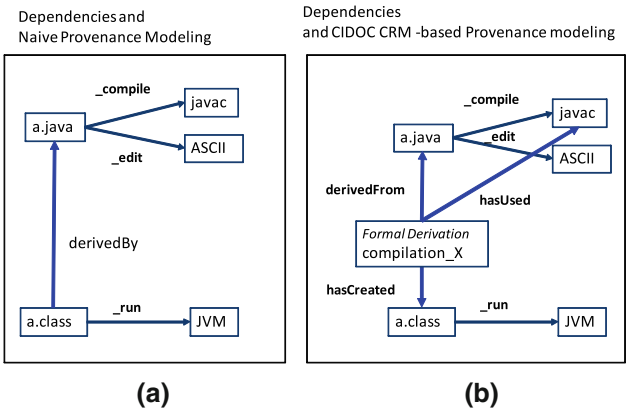

Fig. 6 Dependencies and provenance of digital objects

Figure 7 shows the architecture of ontologies that is used by `PreScan` (presented later in Sect. 7) for capturing the embedded metadata of digital objects. At the top we can see the CIDOC CRM Ontology, at the middle layer the CIDOC CRM Digital Ontology, while at the lowest layer we have the `COD` ontology and other domain specific specializations of it (e.g., the typology of modules shown in Fig. 4).

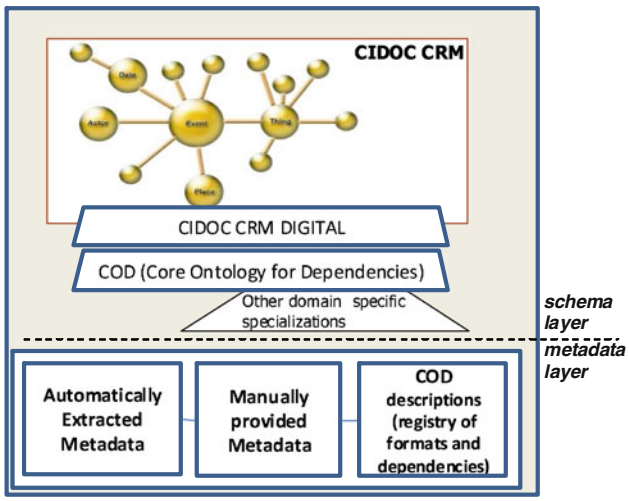

Fig. 7 Architecture of SW ontologies

5 Methodology

Below we sketch a sequence of steps that could be followed by one organization (or digital archivist/curator) for advancing its archive with dependency management services.

St. 1 Identify *intelligibility goals and objectives*.
St. 2 Model the identified goals with *dependency types*. If goals can be hierarchically organized, then this should be reflected in the definition of the dependency types.
St. 3 *Specialize the* `COD` ontology according to the results of the previous step.
St. 4 *Capture the dependencies* of the digital objects of the archive. This can be done manually, automatically or semi-automatically. Tools like the one presented in

Sect. 7 can aid this task. In addition, DC profiles can be defined at this step. The availability of profiles can be exploited for reducing the depth of the dependency graph that has to be captured.

St. 5 Customize, use and *exploit the dependency services* according to the needs. For instance the intelligibility-related services can be articulated with monitoring and notification services.

St. 6 *Evaluate* the services in *real tasks* and *curate* accordingly the repository.

Step 1 strongly depends on the nature of the digital objects and the tasks that we want to perform on them. For instance, suppose we have a set of java files, 2D and 3D images. Common goals for java files include the ability to *compile* and *edit*, while for images they include the ability to render them on screen. The latter could be specialized to two goals, say *2Drendering* and *3Drendering*. At Step 2, we would define five dependency types, say, *compile*, *edit*, *render*, *render2D*, *render3D*, and two subtype relationships: *render2D* $\sqsubseteq$ *render* and *render3D* $\sqsubseteq$ *render*. Step 3 we would extend COD, i.e., we would define *compile* $\sqsubseteq$ *depends*, *edit* $\sqsubseteq$ *depends*, and *render* $\sqsubseteq$ *depends*. The last step (Step 6) concerns evaluation and curation. For instance, suppose the model fails for one particular module, i.e., the consumer of the package is unable to understand the delivered module. Such situations indicate that the recording of dependencies is not complete. For example, suppose a user who cannot run a software component, although the computed gap is empty. This can happen if the component has an additional dependency which has not been recorded. A corrective action would be to add this dependency. Analogously, if a user cannot understand a particular research paper, this is probably because the paper uses concepts or symbols the user cannot understand. These concepts and symbols are actually dependencies which should be recorded. Synopsizing, empirical testing is a useful guide for defining and enriching the dependency graph.

6 Languages for preservation

Here, we elaborate on the various languages that could be used for realizing the aforementioned methodology and scenarios. Specifically, we describe how the various languages and formats (EAST, DEDSL, XFDU, Semantic Web Languages) could be used for the temperature measurements scenario (that was introduced in Section 1), and we discuss their pros and cons.

Suppose we want to preserve files containing temperature measurements from various places on earth. Each file comprises an arbitrary number of lines where each line contains three numerical values corresponding to the longitude, the latitude and the measured temperature (in Celsius degrees). Each line corresponds to the temperature at the coordinate-specified area as it was measured at a certain point in time. The time of measurement is hardwired in the name of the file, e.g., a file named `datafile20080903_12PM.txt` contains measurements taken at 12pm of the 3rd September of 2008. Suppose that the contents of this file are:

25.130 35.325 30.2
25.100 35.161 28.9
25.180 35.333 29.3

We may have several such files (all having the same format though) each one containing measurements at different locations and times.

6.1 Syntax Description

In order to preserve the structure of the file `datafile 20080903_ 12PM.txt` we could make use of the EAST (Enhanced Ada Subse T)[8] language. Each data description record (DDR), according to that language, consists of two packages, one for the logical description and one for the physical description of the data. These two packages are mandatory even if the content of the physical part is empty. The first package includes a logical description of all the described components, their size in bits as well as their location within the set of the described data. The physical part includes a representation of some basic types defined in the logical description and depend on the machine that generates these data: the organization of arrays (i.e., first-index-first, last-index-first) and the bit organization on the medium (high-order-first or low-order-first for big-endian or little endian representation respectively). Figure 8 shows an example of a DDR describing `datafileX.txt`. We defined three different types one for each column of a data file (longitude, latitude , temperature), since each column represents a different kind of data. More precisely the distinction of longitude and latitude is only made because of their different upper and lower bounds.

6.2 Semantic description

We can define semantic descriptions for the entities (longitude, latitude and temperature) of the file `datafileX.txt` aiming at preserving the meaning (clarifying the interpretation) of the terms “longitude” “latitude” and “temperature”. One approach is to use the DEDSL (Data Entity Dictionary Specification Language) [6] language. Figure 9 shows an example of a DEDSL description for `datafileX.txt` according to the implementation of DEDSL using XML. Note that if we have another file with the same kind of information, we could reuse the same semantic descriptions. So semantic descriptions can be considered as reusable modules. These modules can contain abstract data descriptions to which concrete descriptions may refer.

```
package logical_datafileX_description is

type HORIZONTAL_COORDINATE is range -90.00 .. 90.00
for HORIZONTAL_COORDINATE'size 64; --bits

type VERTICAL_COORDINATE is range -180.00 .. 180.00
for VERTICAL_COORDINATE'size 64;

type TEMPERATURE_TYPE is range -100.0 .. 200.0
for TEMPERATURE_TYPE'size 16;

type MEASUREMENT_TUPLE is record
    LONGITUDE:VERTICAL_COORDINATE
    LATITUDE:HORIZONTAL_COORDINATE
    MEASURED_TEMPERATURE:TEMPERATURE_TYPE
end record;
for MEASUREMENT_TUPLE'size use 144;

type MEASUREMENT_BLOCK is array(1..1000) of MEASUREMENT_TUPLE;
for MEASUREMENT_BLOCK'size use 144000;

SOURCE_DATA:MEASUREMENT_BLOCK

end logical_datafileX_description;

package physical_datafileX_description is

end physical_datafileX_description;
```

Fig. 8 An example of an EAST description

6.3 Packaging

Now suppose that we want to preserve and archive a number of datafiles with such measurements. Packaging formats can be used for preparing a package that contains the data files plus their EAST and DEDSL descriptions. We can satisfy such packaging requirements using **XFDU** (XML Formated Data Unit (XFDU) [44,19] which is a standard file format developed by CCSDS (Consultative Committee for Space Data Systems) for packaging and conveying scientific data, aiming at facilitating information transfer and archiving. The benefits of adopting a packaging approach (like that of XFDU) is that we can also add various information about the components of the package. For example, suppose we would like to include information about the user that took the temperatures for each file, as well as the GPS (Global Positioning System) and the thermometer characteristics or the satellite information (if the samplings were made from space). We can easily add the above information using XFDU

```
<?xml version="1.0" encoding=="UTF-8"?>

<DATA_ENTITY_DICTIONARY>
    <DICTIONARY_IDENTIFICATION>
        <DICTIONARY_NAME CASE_SENSITIVITY="NOT_CASE_SENSITIVE">datafileX Dictionary
        </DICTIONARY_NAME>
    </DICTIONARY_IDENTIFICATION>

    <DATA_ENTITY_DEFINITION CLASS="DATA_FIELD" NAME="LONGITUDE">
        <DEFINITIONAL_PART>
            <DEFINITION>
                It represents the longitude for some certain coordinates. Longitudes east of Greenwich shall be designated by the
                use of plus (+) symbol while longitudes west of Greenwich shall be designated with the use of minus (-) symbol.
            </DEFINITION>
            <SHORT_DEFINITION>Longitude</SHORT_DEFINITION>
            <UNITS>deg</UNITS>
        </DEFINITIONAL_PART>
        <REPRESENTATIONAL_PART DATA_TYPE="REAL">
            <RANGE MIN="-180.00" MAX="+180.00"/>
        </REPRESENTATIONAL_PART>
    </DATA_ENTITY_DEFINITION>
    <DATA_ENTITY_DEFINITION CLASS="DATA_FIELD" NAME="LATITUDE">
        <DEFINITIONAL_PART>
            <DEFINIITION>
                It represents the latitude for some certain coordinates.  Latitudes north of the Equator shall be designated by the
                use of plus (+) symbol while latitudes south of the Equator will be designated by the use of minus (-) symbol.
            </DEFINITION>
            <SHORT_DEFINITION>Latitude</SHORT_DEFINITION>
            <UNITS>deg</UNITS>
        </DEFINITIONAL_PART>
        <REPRESENTATIONAL_PART DATA_TYPE="REAL">
            <RANGE MIN="-90.00" MAX="+90.00"/>
        </REPRESENTATIONAL_PART>
    </DATA_ENTITY_DEFINITION>
    <DATA_ENTITY_DEFINITION CLASS="DATA_FIELD" NAME="TEMPERATURE">
        <DEFINITIONAL_PART>
            <DEFINITION>
                It represent the temperature in the area with
                the specific coordinates.
            </DEFINITION>
            <SHORT_DEFINITION>Temperature</SHORT_DEFINITION>
            <UNITS>Celsius degrees</UNITS>
        </DEFINITIONAL_PART>
        <REPRESENTATIONAL_PART DATA_TYPE="REAL">
            <RANGE MIN="-100.0" MAX="200.0"/>
        </REPRESENTATIONAL_PART>
    </DATA_ENTITY_DEFINITION>
</DATA_ENTITY_DICTIONARY>
```

Fig. 9 An example of a DEDSL description

```
<?xml vesrion="1.0" encoding="UTF-8"?>

<xfdu:XFDU xmlns:xfdu="http://www.ccsds.org">
  <packageHeader ID="id"/>
  <informationPackageMap ID="id">
    <xfdu:contentUnit ID="file:/avgCalculator.java">
    <dataObjectPointer dataObjectId="dataObjectAC"/>
      <xfdu:contentUnit ID="file:/data">
        <dataObjectPointer dataObjectID="dataObject20080903_12PM"/>
        <dataObjectPointer dataObjectID="dataObject20080903_12AM"/>
      </xfdu:contentUnit>
     </xfdu:contentUnit>
  </informationPackageMap>
  <dataObjectSection>
    <dataObject size="5620" ID="dataObjectAC">
       <byteStream size="5620" mimeType="text/plain">
          <fileLocation href="file:/avgCalculator.java"/>
       </byteStream>
    </dataObject>
      <dataObject size="122108" ID="dataObject20080903_12PM">
        <byteStream size="122108" mimeType="text/plain">
          <fileLocation href="file:/data/datafile20080903_12PM.txt"/>
        </byteStream>
      </dataObject>
      <dataObject size="134554" ID="dataObject20080903_12AM">
        <byteStream size="134554" mimeType="text/plain">
          <fileLocation href="file:/data/datafile20080903_12AM.txt"/>
        </byteStream>
      </dataObject>
  </dataObjectSection>
</xfdu:XFDU>
```

Fig. 10 An example of a XFDU package

since we just have to add the necessary information in the package. For instance, we could describe such information using CIDOC CRM ontology. Such descriptions could be expressed in XML format (e.g., CIDOC CRM CORE) or as an RDF/XML file[4] (the RDF-case is described in more detail in Sect. 6.4). In both cases the file could be included in the package. It is obvious that the major benefit from the use of XFDU is that we can package together heterogenous modules (java programs, datafiles, GPS info, provenance data) and deliver them to the user, or archive them, as a single (ideally self-describing) unit.

As another example suppose that we have a java program that calculates the daily, weekly, and monthly average temperatures of various locations. To make such aggregate calculations the program needs a number of data files. For example, to calculate the average temperature of each location during the 3rd of September of 2008, we need all data files with names datafile20080903_ ∗ ∗ ∗ ∗.txt. For delivering this program to a user, as an application that contains past measurements, we have to pack together the required data files. The resulting XFDU package will contain both the java program and the data files. Figure 10 shows an example of such a package (for reasons of brevity we depict only two data files), while Fig. 11 illustrates the structure of such a package.

[4] That uses the classes and properties defined in http://cidoc.ics.forth.gr/rdfs/caspar/cidoc.rdfs#.

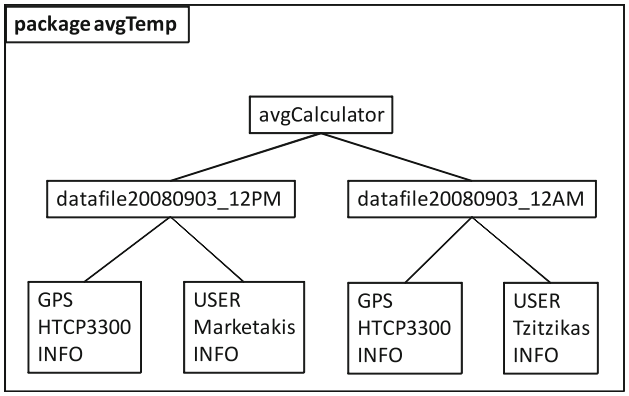

Fig. 11 Structure organization for a XFDU package

```
<?xml version='1.0'?>
<rdf:RDF xml:lang="en"
         xmlns:rdf="http://www.w3c.org/1999/02/22-rdf-syntax-ns#"
         xmlns:rdfs="http://www.w3c.org/2000/01/rdf-schema#">

    <rdfs:Class rdf:ID="Sample"/>
    <rdf:Property rdf:ID="Longitude">
        <rdfs:domain rdf:resource="#Sample"/>
        <rdfs:range rdf:resources="&rdfs;Literal"/>
    </rdf/Property>
    <rdf:Property rdf:ID="Latitude">
        <rdfs:domain rdf:resource="#Sample"/>
        <rdfs:range rdf:resource="&rdfs;Literal"/>
    </rdf:Property>
    <rdf:Property rdf:ID="Temperature">
        <rdfs:domain rdf:resource="#Sample"/>
        <rdfs:range rdf:resource="&rdfs;Literal"/>
    </rdfs:Property>
</rdf:RDF>
```

Fig. 12 Example of an ontology for measurements expressed in RDF/S

6.4 Semantic Web Languages

Here, we describe an alternative approach that relies on Semantic Web (SW) languages. The adoption of SW languages (like RDF/S) has an additional benefit. A top/upper level ontology can be used to describe (syntactically and semantically) the form of the data and other files can instantiate this ontology. These instantiations are actually the data themselves. When creating a new data file, there is no need to create its DEDSL or EAST description every time. Instead, we have to represent the data using the syntax specified by the ontology. Moreover, whenever we want to preserve more data types we can extend that ontology. For example, past data files can contain only the longitude, the latitude and the temperature, while current ones may contain also the name of each location, or the thermometer used for the measurement. In such cases, two different kinds of DEDSL/EAST descriptions have to be created and used. In the SW approach, we just have to extend the top ontology. Figure 12 shows an indicative ontology for our running example expressed in RDF/S XML. Other ontologies of wider scope can be used as well (e.g., CIDOC CRM).

Another benefit of using SW languages is that data and their descriptions are tightly coupled. In contrast, EAST/DEDSL-descriptions are represented as separate files and for this reason packaging formats are important. On the other hand, in RDF, a data file would itself define the data type of each data element in the file. To clarify this point consider the following line from our running example: `25.130 35.325 30.2`. This line does not provide any information regarding what `25.130` might be, it could be either the longitude, the latitude or the temperature. On the other hand its RDF representation would be:

```
<temperature:Sample
 rdf:about=''sampingId20080903_12PM_
                35.233_25.343>
 <temperature:Longitude= ``25.130''/>
 <temperature:Latitude= ``35.325''/>
 <temperature:Temperature= ``30.2''/>
</temperature:Sample>
```

This part of RDF can be understood without the existence of any other (EAST/DEDSL) file.

7 Implementations

Here we detail the functionality and architecture of a tool named `GapMgr` that realizes all services described previously. It comprises:

(a) `GapMgr` API. This is a programmatic interface written in Java. Instead of directly adopting one of the existing APIs and main memory models of the Semantic Web, we designed a new one focusing only on the requirements of the problem at hand.
(b) Two different implementations of the API. The first is a main memory implementation. The persistence layer is a plain file-system based. The second is an implementation over the SWKM (Semantic Web Knowledge Middleware).[5]
(c) An end-user Web-based application developed using GWT (Google Web Toolkit).[6] It has a modular design and it can work with both implementations of the API. Figure 13 shows the Use Case Diagram of this application.

7.1 Web-based GUI

The graphical user interface has been implemented using GWT offering an easy to use Web-based UI. Several partners (both technical and data providers) of the CASPAR project, including University of Leeds, CNRS,[7] CIANT,[8] British Atmospheric Data Centre (UK), European Space Agency (ESA-ESRIN), have already deployed it and have started using it.[9] Figures 14 and 15 show some indicative screendumps; The former for defining dependencies and the latter for computing intelligibility-aware packages. It can be considered as a semantic registry for preservation.

7.2 Experimental evaluation

7.2.1 Real data sets

To investigate whether `COD` can capture (as it is or by extending it) modules and dependencies from various different domains, we used it for expressing the contents of various existing collections. We have successfully represented the following collections.

– *File formats and software products*
PRONOM [34] is an online registry of technical information. It provides information about file formats, software products and other technical components that are required to support long-term access to electronic records and other digital objects of cultural, historical or business value. Currently approximately 850 records are listed. All have been extracted and loaded to `GapMgr`. However, only a few dependencies are defined between these formats.
– *Modules and dependencies from the CASPAR project*
We have imported several modules along with their dependencies in the context of the CASPAR project. In addition, we defined several profiles for the various communities involved. The resulting data set contains modules from the cultural domain (UNESCO sites) and contemporary arts domain (mainly coming from CNRS, INA,[10] University of Leeds, and CIANT). Furthermore, there are more than 1,200 modules exported from a Registry of formats from the scientific domain.

7.2.2 Synthetic data sets

To evaluate the efficiency of the two implementations, we created several synthetic data sets. We developed a generator that takes as input two parameters: the number of modules

[5] http://athena.ics.forth.gr:9090/SWKM/.
[6] http://code.google.com/webtoolkit/.
[7] Centre National de la Recherche Scientifique, France.
[8] International Centre for Art and New Technologies, Prague, Czech Republic.
[9] http://developers.casparpreserves.eu:8080/CasparGui/. http://139.91.183.30:3025/GapManagerGWT_SWKM/.
[10] Institut National de l'Audiovisuel, France.

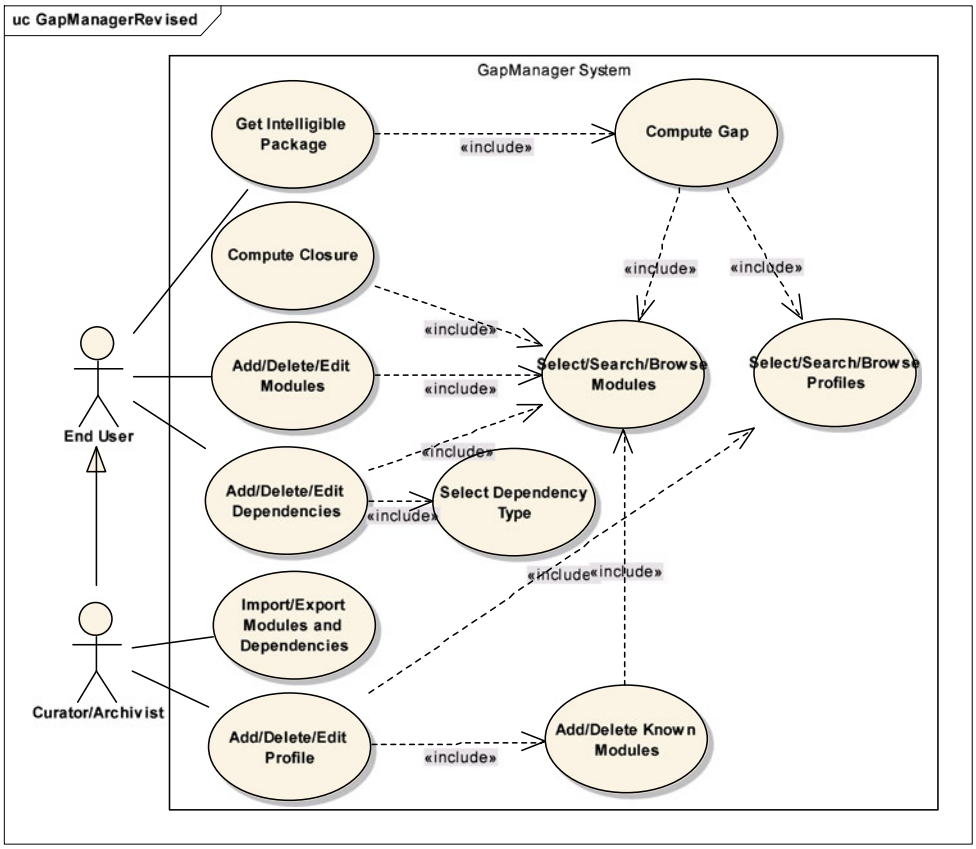

Fig. 13 The use case diagram of `GapMgr`

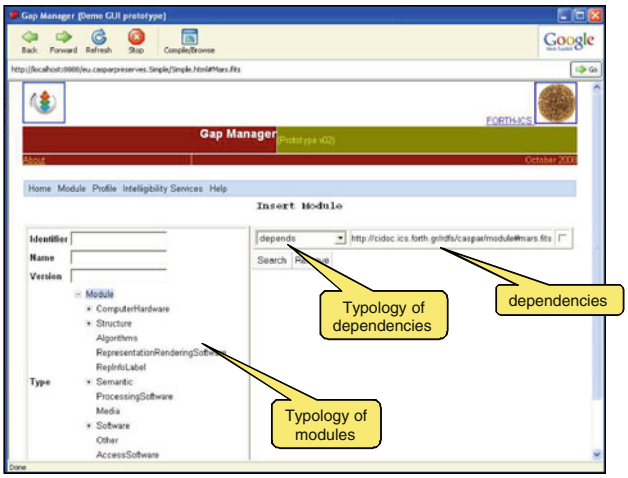

Fig. 14 The GUI of `GapMgr`: defining dependencies

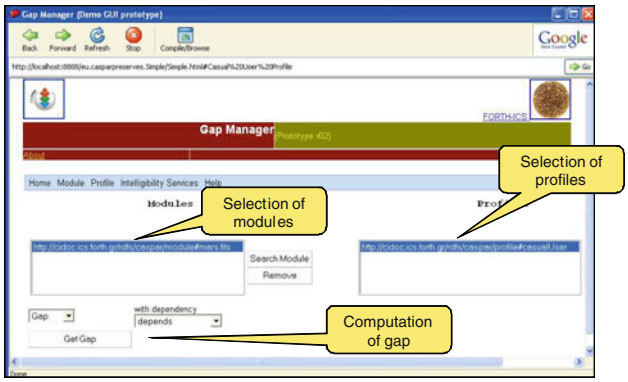

Fig. 15 The GUI of `GapMgr`: computation of the intelligibility gap

N to be created, and the "density" of the dependency graph (two options are supported: *sparse* and *dense*). The generator proceeds as follows: at first it creates N modules. Then it creates random dependencies among these modules. If the option is set to "sparse" then it creates $N \log N$ dependencies, otherwise (i.e., if "dense") it creates $2N \log N$ dependencies. Subsequently $\log N$ profiles are defined.

To each of these profiles a random set of modules is associated in the range of $[1 \ldots \sqrt{N}]$. The average depth of the obtained dependency graph as well as the average size of the closure of a module are shown in Table 4. We can see that the average depth of the graph remains the same even if the number of modules increases. On the other hand the average size of the closure increases. This means that as the number of modules grows, the dependency graph becomes broader

Table 4 Features of the synthetic dependency graphs

N	10^3	10^4	10^5
Average depth	9	10	10
Average closure size	261	2174	17227
Max depth	23	24	25
Max closure size	675	5930	50838

rather than deeper, and this is due to the adopted generation method.

7.2.3 Measured tasks (description and algorithms)

We measured the time requirements for the following tasks:

- *Computing a closure* (i.e., $Closure(t)$ or $Closure(S)$)
  We traverse the dependency graph and collect all dependent modules starting from t or S. The complexity of this task depends on the number of modules but mainly on the density of the dependency graph (the more dense the graph is, the more modules the closure will contain).
- *Computing a gap* (i.e., $Gap(t, u)$)
  The computation of the gap between one module t and one profile u is more complex than the computation of the closure of a single module, since here we have to compute the closures of all modules contained in the profile. Consequently, the complexity of this task depends on the size of the profile and the density of the dependency graph.
- *Deciding intelligibility* (i.e., $isIntelligible(t, u)$)
  To decide whether a module t is intelligible by a profile u, we have to compute the closure of all modules in u. If t or its direct dependencies exist in the closure set, then this module is intelligible by the profile.
- *Deciding dependency* (i.e., $depends(t, t')$)
  Module t depends (directly or indirectly) on t' if t' belongs to the closure of t (i.e., if $t' \in Nr^+(t)$). Instead of computing the entire closure of t, we can compute it gradually and terminate whenever t' is about to be added to the closure. It follows that this task is faster than the computation of the entire closure.
- *Adding a dependency*
  Before adding a dependency $t > t'$, we have to guarantee that this addition will not create a cycle. For this reason we first check whether $t' >^+ t$ holds, and if yes we reject the request. Therefore, the cost of this task is roughly equal to the cost of $depends(t, t')$.
- *Deleting a module*
  Before deleting a module t we have to remove all dependencies that involve t. In addition, we have to "curate" the profiles, i.e., for each profile u such that $t \in Nr^*(T(u))$, we have to replace t with the modules $Nr(t)$. At the end, we delete t.
- *Upgrading a module*
  A new module t_{new} is created with the same dependencies with the specified module t (i.e., $Nr(t_{new}) = Nr(t)$).

7.2.4 Implementation settings

We performed measurements over the following settings:

(A) We used the dedicated main memory API (in Java) that we have developed.
(B) We used the Semantic Web Knowledge Middleware. In this case, modules, dependencies and profiles are stored in the SWKM repository. SWKM offers a wide and scalable suite of basic services for validating, storing, querying, updating and exporting descriptive metadata expressed in RDF/S. All services are based on a common knowledge repository enabling the consistency of its contents. It also offers knowledge evolution services (declarative update languages, comparison services and versioning). In this setting all measurements have been done by sending read/update queries (in RQL[16] and RUL[20]) to the repository through the SWKM WS client.

A deployment diagram of the architecture of the entire system (GapManager and associated components) is shown in Fig. 16.

7.2.5 Results

Table 5 reports execution times for various numbers of modules. If the number of modules is low (e.g., 1000) all tasks are performed very fast. As the number of modules increases the time to perform some tasks, especially those that require traversals, increases. For example, for $|\mathcal{T}| = 10^5$, the computation of gap takes around 5 s.

7.2.6 Implementation over SWKM

With the main memory implementation of the API, all modules, dependencies and profiles are loaded in main memory. This approach is proved efficient, however, if the volume of data increases then they may not fit in memory. The implementation over SWKM overcomes such limitations, as all data are stored in the SWKM repository and all services are implemented by appropriate calls to the declarative query and update language (RQL and RUL respectively). However, and regarding efficiency, this approach has the overhead of parsing the results of the queries which are delivered in RDF/XML format. For big results, the delay is unacceptably long. For instance, the execution of a query comprising of 2,300 modules takes about 5 s to compute and get (through the

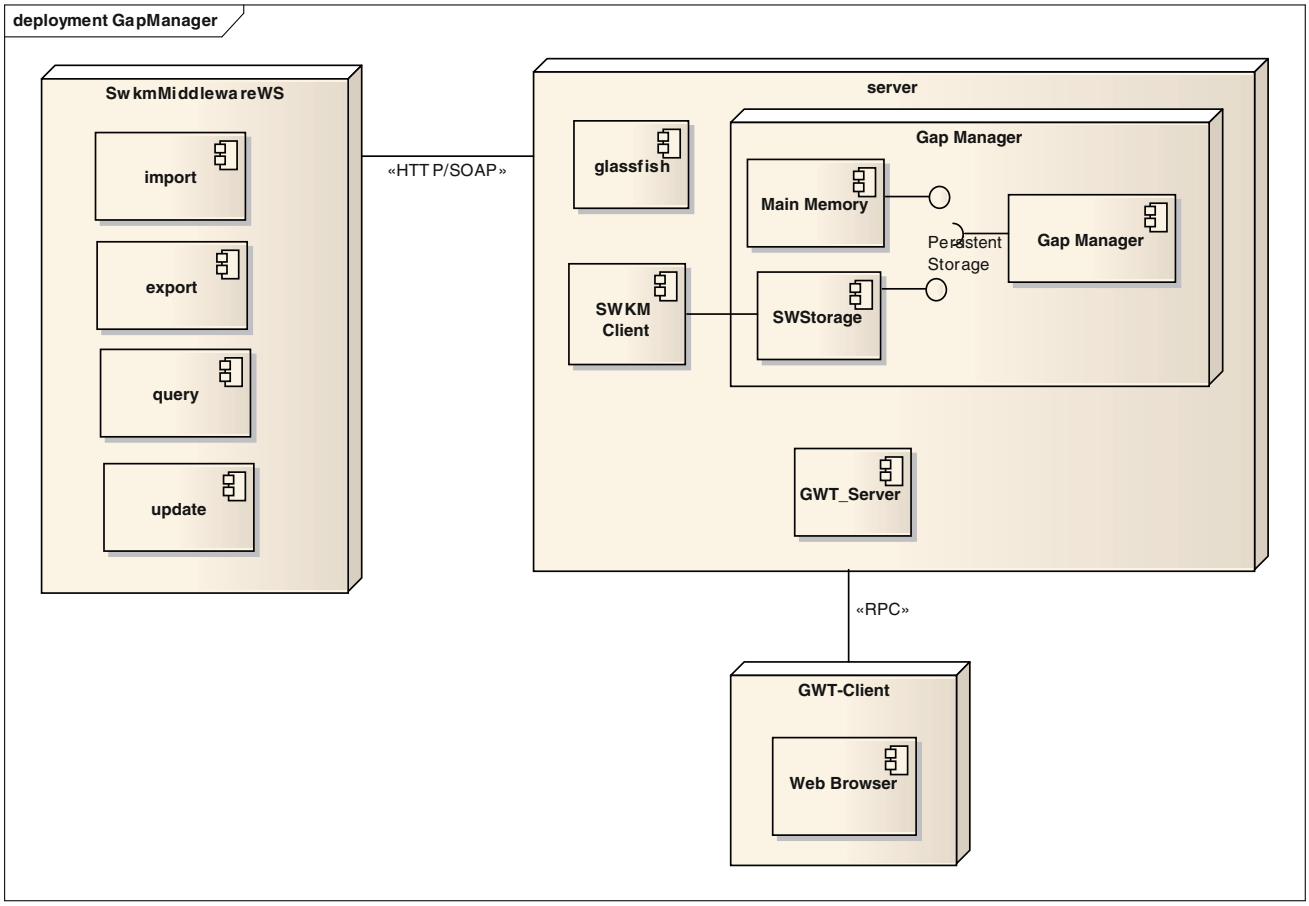

Fig. 16 GapMgr architecture

Web Service), while the parsing of the results takes approximately 50 s. The problem could be alleviated by extending the implementation of RQL so that to support result sets and cursors. Finally, a general approach to speedup the evaluation of SW queries it to adopt a grid computing infrastructure. For instance [31] reports results of querying over 300 million SW triples using Openlink Virtuoso.[11]

7.3 Automating the ingestion of digital objects

Recently, we developed a tool that can automate the ingestion of metadata. Specifically, `PreScan`[12] [22] scans entire file systems, automatically extracts the embedded metadata of the encountered digital files, and allows the addition of user-provided metadata or extra dependencies. It uses external metadata extractors (currently JHOVE.[13]) and exploits the repository of `GapMgr` for modeling and recording dependencies. The benefit of adopting a repository is that the extracted metadata are linked with their format descriptions. In case the automatically extracted dependencies are not adequate, users can add extra dependencies using `GapMgr`.

Table 5 Execution times using MM API

Services	Time(ms)		
	$N = 10^3$	$N = 10^4$	$N = 10^5$
getClosure(t)	2	51	820
depends(t,t′)	1	33	426
addDependency(t,t′)	1	33	411
getDirectDependencies(t)	0	0	0
getDirectDependents(t)	0	0	0
isIntelligible(t,u)	8	212	4631
deleteModule(t)	0	1	1
Gap(t,u)	13	223	5412
upgradeModule(t)	1	2	2

[11] http://www.openlinksw.com/virtuoso/.

[12] http://www.ics.forth.gr/prescan/.

[13] http://hul.harvard.edu/jhove/.

Table 6 Dependency management in other domains

Work	Modules	Assumed goal (when recording dependencies)	Types of dependencies (between modules)	Reason why dependencies are recorded
[2]	Software components	To install or uninstall a composite component	Mandatory, optional, negative	To reason on istallability, deinstallability
[10]	Software components	Achieve goals, satisfy soft goals, complete tasks, provide and consume resources	Goal, task, resource, soft goal	To aid the selection of the most appropriate component
[41]	Software components	Ability to compile/run expressing which component affects the behavior of other components	The dependencies of a component are categorized to (a) internal (i.e., intra-component), and (b) external (inter-component). The internal ones are further categorized to (a1) implementation-based and (a2) operation-based. The external ones are further distinguished to (b1) hardware, (b2) software (i.e., required interfaces), and (b3) causal	To support the process of evolution and testing in component-based systems
[33]	Ontologies	Considers the dependencies in the ontology representations (reuse/extend inter-ontology relationships)	Isa, reference	To aid the development of ontologies, in particular when changes occur, i.e., to address questions of the form: if an ontology changes what should happen in the dependent ontologies?
Maven	Software components	Software engineering	Compile, provided, runtime, test, system, import	Build, compile, and the like

8 Related work

At first, we should say that in comparison to existing registries for preservation (like PRONOM), our approach has the following key advantages: (a) typed dependencies (between modules), (b) typed modules, (c) the notion of DC profiles (for making the assumptions explicit and enabling profile-aware packaging).

Regarding dependencies, we should note that dependency management is an important requirement that is subject of research in several traditional or more recent areas: from Software Engineering (at [42,41,43,2]), to Ontology Engineering (at [15,33]). In software engineering the various build tools (e.g., make, gnumake, nmake, jam, ant) are related to our problem, since they allow defining dependencies and those tasks required to be performed in order to build a software project. In ontology, engineering an analogous problem is how to reflect a change of an ontology to the dependent ontologies (i.e., to those that reuse or extend parts of it), which may be stored in different sites. Another related problem is the schema evolution problem, i.e., the problem of reflecting schema changes to the underlying instances. Actually, this problem is related to the evolution of modules and dependencies.

Table 6 lists and describes in brief a number of dependency management approaches that have been described in the literature: [10] focuses on dependency types to enable the selection of the best software architecture, [41] analyzes the dependencies in large scale component-based systems to ensure the runability and compilability, while [2] concentrates on the success of the deployment and the safety of the system. Finally, [33] elaborates on the dependencies between ontologies and distinguishes dependencies to *super-sub* and *referring-to* dependencies.

As we can see there is much heterogeneity on the types of modules, the kinds of goals (that determine the semantics of a dependency), the types of dependencies and the dependency management services. Probably, in each preservation application domain, we have to model the corresponding modules and dependency types and identify the needed services. It does not seem that we could have one single modeling for all cases. This observation justifies the selection of SW languages (due to the extensibility they offer).

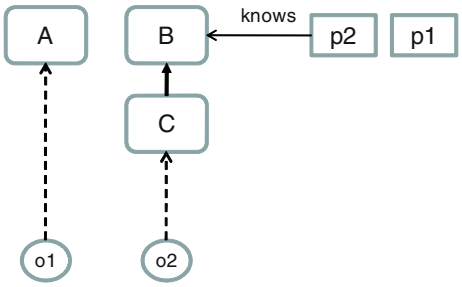

Fig. 17 Example of an ontology gap

9 Extensions of the model

In this section, we discuss other kinds of gaps, including gaps that concern *descriptive metadata*, gaps in the form of *change operations*, and gaps of *finer granularity*.

So far, we have focused mainly on the *representation information* (OAIS RepInfo) dependencies of digital objects. Let us now discuss the dependencies of digital objects from the *descriptive metadata* point of view. Recall that according to OAIS, an object can have various descriptive metadata. Let us assume that all these metadata are represented with respect to ontologies expressed in RDFS. In particular, consider two objects o_1 and o_2 where the metadata of o_1 are expressed with respect to an ontology A, while those of o_2 are expressed with respect to an ontology C. Now consider a particular community p_1 that is not familiar with any of the ontologies used for expressing metadata, and a community p_2 that is familiar with ontology B and suppose that C is a specialization of B (i.e., it reuses and extends elements of B), as shown in Fig. 17. Familiarity with an ontology means familiarity with the domain of the ontology and the conceptualization of that ontology. We could define the *descriptive gap* between an object o and a community profile p, denoted by $dgap(o, p)$, as the set of ontologies that a member of the p community needs to understand in order to understand the metadata of o. In our case, this would mean that:

$$dgap(o_1, p_1) = A$$
$$dgap(o_2, p_1) = C \cup B$$
$$dgap(o_1, p_2) = A$$
$$dgap(o_2, p_2) = C$$

Furthermore, we can refine the granularity of modules: instead of considering ontologies as modules, we can consider the elements of these ontologies as modules. To this end we could exploit *comparison operators*, else called *diff* or *delta* (Δ) operators, like those proposed in [24,45], as well as methods that take as input the set of elementary change operations that is returned by a comparison operator, and output an equivalent set of high level changes [25]. For example, consider the case illustrated in Fig. 18. If we assume that equality of concept names implies equality of concepts, we could define:

$$dgap(o_1, p_1) = \Delta(A \rightarrow A) = \emptyset$$
$$dgap(o_2, p_1) = \Delta(A \rightarrow C \cup B)$$
$$dgap(o_1, p_2) = \Delta(B \rightarrow A)$$
$$dgap(o_2, p_2) = \Delta(B \rightarrow C \cup B)$$

According to this view, a gap comprises change operations. A detailed treatment of such cases goes beyond the scope of this article (for more see [45]).

An alternative approach to define fine grained gaps that consist of modules (not change operations) is also possible. The key observation is that *instanceOf* and *isA* relations are special kinds of dependencies, actually they carry more meaning than a plain dependency relation. This means that an *isA* hierarchy could be construed as a dependency graph in our framework (where each subclass depends on its superclasses). In the example of Fig. 18, this means that we have the dependencies $o_2 <_{instanceOf}$ `Student` $<_{isA}$ `Person`. Under this perspective, $Gap(o_2, p_2) =$ {`Student`}. It follows that according to this view, profiles, as well as intelligibility gaps, can contain all kinds of RDF elements.

So far we have considered intelligibility gaps that comprise sets of modules. The dependency types that participate to the computation of gap could also be returned as they convey extra meaning which could be exploited and recorded (e.g., in the manifest file of XFDU). For instance, we can define gaps as sets of paths where a path is a sequence of $(depType, module)$ pairs, or RDF triples of the form $(subject, predicate, object)$, indicating the specializations of the ontology that are required. In the example of Fig. 18, where
$Gap(o_2, p_2) =$ {`Student`}, a more informative gap would be:
$IGap(o_2, p_2) =$ {`instOf Student subclassOf Person`}. In a triple form we could write:

$$IGap(o_2, p_2) = \{\{o_2 \texttt{ instOf Student}\},$$
$$\{\texttt{Student subclassOf Person}\}\}$$

10 Concluding remarks

This article described a methodology for representing intelligibility-related dependencies and exploiting them for various preservation tasks. The described services and systems can be useful not only for archives that follow a SW-based preservation approach, but also for those that are based on conventional approaches (like EAST, DEDSL, XFDU, SAFE). To summarize, we could say that dependency management can be exploited for:

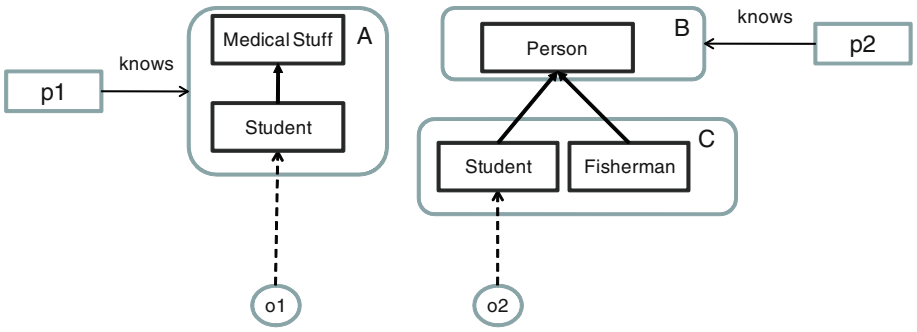

Fig. 18 Example of a more refined ontology gap

– *Enabling a disciplined method for deciding what metadata to include in a package*.
It allows deriving packages that are intelligible for a certain community and the assumed knowledge of that community is explicitly specified (in the form of a profile). For example if we want to derive a package for a community that is not familiar with EAST/DEDSL, then apart from the EAST/DEDSL files for the data files we should also include the descriptions of these standards. The latter could be a set of documents in the `pdf` format (if of course that format is supposed to be known by that community).
– *Supporting the curation of existing packages.*
The services could aid protecting archives from obsolescence risks. For example, if a digital file, e.g., `a.east` is identified as problematic (or if EAST becomes obsolete), then the repository of a tool like `GapMgr` could be exploited for identifying the packages that have to be revised.
– *Enabling intelligibility-aware packaging also for the SW approach.*
Recall that ontologies may reuse/extend elements coming from other ontologies. For example, if the assumed designated community is supposed to know CIDOC CRM ontology, then for constructing packages containing metadata expressed with respect to a specialization of CIDOC CRM, we are not obliged to include the CIDOC CRM ontology (i.e., its representation in RDF) in the package, but only its specialization (if that specialization is not member of the designated profile too).
For the case where the profile is a version of CIDOC CRM and the object is described with respect to a newer version of CIDOC CRM, then the comparison functions introduced in [45] could also be considered as a way to fill the intelligibility gap.

Issues that are worth further research include: (a) adoption of rules (for capturing more complex cases of dependencies, or for defining the properties of dependencies), (b) investigation of the benefits/applicability of metamodeling approaches, (c) investigation of relationships with formal ontology, and (d) automatic extraction of dependencies.

Acknowledgements Motivation for this work is the ongoing EU project CASPAR (FP6-2005-IST-033572) whose objective is to build a pioneering framework to support the end-to-end preservation lifecycle for scientific, artistic and cultural information based on existing and emerging standards.

References

1. CASPAR (Cultural, Artistic and Scientific knowledge for Preservation, Access and Retrieval). FP6- 2005-IST-033572. http://www.casparpreserves.eu/
2. Belguidoum, M., Dagnat, F.: Dependency Management in Software Component Deployment. Electronic Notes in Theoretical Computer Science **182**, 17–32 (2007)
3. Brickley, D., Guha, R.V.: Resource Description Framework (RDF) Schema Specification: Proposed Recommendation, W3C, March 1999. http://www.w3.org/TR/1999/PR-rdf-schema-19990303
4. Cheney, J., Lagoze, C., Botticelli, P.: Towards a theory of information preservation. In: Proceedings of the 5th European Conference on Research and Advanced Technology for Digital Libraries, pp. 340–351, ECDL'01, London, UK. Springer-Verlag, Berlin (2001)
5. Cooper, B.F., Garcia-Molina, H.: InfoMonitor: unobtrusively archiving a World Wide Web server. Int. J. Digit. Libr. **5**(2), 106–119 (2005)
6. DEDSL Language. http://east.cnes.fr/english/index.html
7. Day, M.: Integrating metadata schema registries with digital preservation systems to support interoperability: a proposal. In: Proceedingss of DC 2003. Supporting Communities of Discourse and Practice-Metadata Research & Applications, vol. 2, Seattle, Washington (USA), September (2003)
8. EAST Language. http://east.cnes.fr/english/page_east.html
9. Ferreira, M., Baptista, A.A., Ramalho, J.C.: An intelligent decision support system for digital preservation. Int. J. Digit. Libr. **6**(4), 295–304 (2007)
10. Franch, X., Maiden, N.A.M.: Modeling component dependencies to inform their selection. In: Proceedings of the 2nd International Conference on COTS-Based Software Systems, ICCBSS'03, Ottawa, Canada, February, 2003. Springer, Berlin (2003)
11. Hedstrom, M.: Digital preservation: a time bomb for digital libraries. Comput. Hum. **31**(3), 189–202 (1997)
12. Hunter, J., Choudhury, S.: A semi-automated digital preservation system based on semantic web services. In: Proceedings of the 4th ACM/IEEE-CS Joint Conference on Digital Libraries, JCDL'04, pp. 269–278, New York, NY, USA. ACM Press, New Yoek (2004)

13. Hunter, J., Choudhury, S.: PANIC: an integrated approach to the preservation of composite digital objects using Semantic Web services. Int. J. Digit. Libr. **6**(2), 174–183 (2006)
14. International Organization For Standardization (OAIS): Open Archival Information System—Reference Model. Ref. No ISO 14721:2003 (2003)
15. Jarrar, M., Meersman, R.: Formal ontology engineering in the DOGMA approach. In: Proceedings of the International Conference on Ontologies, Databases and Applications of Semantics, ODBase'02, pp. 1238–1254. Springer, Berlin (2002)
16. Karvounarakis, G., Christophides, V., Plexousakis, D.: Querying Semistructured (Meta)data and Schemas on the Web: The case of RDF & RDFS. Technical Report 269. ICS-FORTH, Heraklion (2000). (http://www.ics.forth.gr/proj/isst/RDF/rdfquerying.pdf)
17. Lee, K.H., Slattery, O., Lu, R., Tang, X., McCrary, V.: The state of the art and practice in digital preservation. J. Res. Natl. Inst. Stand. Technol. **107**(1), 93–106 (2002)
18. Lin, C.H., Hong, J.S., Doerr, M.: Issues in an inference platform for generating deductive knowledge: a case study in cultural heritage digital libraries using the CIDOC CRM. Int. J. Digit. Libr. **8**(2), 115–132 (2008)
19. Lucas, A.: XFDU packaging contribution to an implementation of the OAIS reference model. In: Proceedings of the International Conference, PV'2007. Ensuring the Long-Term Preservation and Value Adding to Scientific and Technical Data, Edinburgh, November (2005)
20. Magiridou, M., Sahtouris, S., Christophides, V., Koubarakis, M.: RUL: A declarative update language for RDF. In: Proceedingss of the 4th International Conference on the Semantic Web, ISWC-2005, Galway, Ireland, November (2005)
21. Maniatis, P., Roussopoulos, M., Giuli, T.J., Rosenthal, D.S.H., Baker, M.: The LOCKSS peer-to-peer digital preservation system. ACM Trans. Comput. Syst. (TOCS) **23**(1), 2–50 (2005)
22. Marketakis, Y., Tzanakis, M., Tzitzikas, Y.: Prescan: towards automating the preservation of digital objects. In: Proceedings of the International ACM Conference on Management of Emergent Digital Ecosystems, MEDES'09, pp. 404–411, Lyon, France, October (2009)
23. Nelson M.L., McCown F., Smith J.A., Klein M.: Using the web infrastructure to preserve web pages. Int. J. Digit. Libr. **6**(4):327–349 (2007)
24. Noy, N.F., Musen, M.A.: PromptDiff: A fixed-point algorithm for comparing ontology versions. In: Proceedings of the 18th National Conference on Artificial Intelligence, AAAI-2002, pp. 744–750, Edmonton, Alberta, (2002)
25. Papavassiliou, V., Flouris, G., Fundulaki, I., Kotzinos, D., Christophides, V.: On detecting high-level changes in RDF/S KBs. In: Proceedings of the 8th International Semantic Web Conference, ISWC 2009, October 2009
26. PLANETS: Digital Preservation Research and Technology. HPRN-CT-2002-00308. http://www.planets-project.eu
27. Rajasekar, A., Moore, R., Berman, F., Schottlaender, B.: Digital preservation lifecycle management for multi-media collections. In: Proceedings of the 8th International Conference on Asian Digital Libraries, ICADL, vol. 3815, pp. 380–384, December (2005)
28. Rauch, C., Rauber, A.: Preserving digital media: towards a preservation solution evaluation metric. In: Proceedings of the 7th International Conference on Asian Digital Libraries, ICADL'04, pp. 203–212, Shanghai, China, (2004)
29. Rauch, C., Franz, P., Strodl, S., Rauber, A.: Evaluating preservation strategies for audio and video files. In: Proceedings of the DELOS Digital Repositories Workshop, Heraklion, Crete, Greece, (2005)
30. Ross, S., Hedstrom, M.: Preservation research and sustainable digital libraries. Int. J. Digit. Libr. **5**(4), 317–324 (2005)
31. Stenzhorn, H., Srinivas, K., Samwald, M., Ruttenberg, A.: Simplifying access to large-scale health care and life sciences datasets. Lect. Notes Comput. Sci. **5021**, 864–868 (2008)
32. Strodl, S., Becker, C., Neumayer, R., Rauber, A.: How to choose a digital preservation strategy: evaluating a preservation planning procedure. In: Proceedings of the 2007 conference on Digital libraries, pp. 29–38. ACM Press, New York (2007)
33. Sunagawa, E., Kozaki, K., Kitamura, Y., Mizoguchi, R.: An environment for distributed ontology development based on dependency management. In: Proceedings of the 2nd International Semantic Web Conference, ISWC'03, pp. 453–468. Springer, Berlin (2003)
34. The Technical Registry PRONOM (The National Archives). (http://www.nationalarchives.gov.uk/pronom)
35. Theodoridou, M., Tzitzikas, Y., Doerr, M., Marketakis, Y., Melessanakis, V.: Modeling and querying provenance by extending using CIDOC CRM. J Distrib. Parallel Databases **27**, 169–210 (2010)
36. Thibodeau, K.: Overview of technological approaches to digital preservation and challenges in coming years. Council on Library and Information Resources (CLIR). The State of Digital preservation: An International Perspective, April (2002)
37. Tzitzikas, Y.: Dependency Management for the preservation of digital information. In: Proceedings of the 18th International Conference on Database and Expert Systems Applications, DEXA'07, Regensburg, Germany, September 2007. Springer-Verlag, Berlin (2007)
38. Tzitzikas, Y.: On preserving the intelligibility of digital objects through dependency management. In: Proceedings of the International Conference PV'2007. Ensuring the Long-Term Preservation and Value Adding to Scientific and Technical Data, Oberpfaffenhofen, Munich, October (2007)
39. Tzitzikas, Y., Flouris, G.: Mind the (intelligibily) gap. In: Proceedings of the 11th European Conference on Research and Advanced Technology for Digital Libraries, ECDL'07, Budapest, Hungary, September 2007. Springer-Verlag, Berlin (2007)
40. Tzitzikas, Y., Kotzinos, D., Theoharis, Y.: On ranking RDF schema elements (and its application in visualization). J. Univ. Comput. Sci. **13**(12), 1854–1880 (2007)
41. Vieira, M., Richardson, D.: Analyzing dependencies in large component-based systems. In: Proceedings of the 17th IEEE International Conference on Automated Service Engineering, ASE'02. IEEE Computer Society, Los Alamitos (2002)
42. Vieira, M., Dias, M., Richardson, D.J.: Describing dependencies in component access points. In: Proceedings of the 23rd International Conference on Software Engineering, ICSE'01, pp. 115–118, Toronto, Canada, (2001)
43. Walter, M., Trinitis, C., Karl, W.: OpenSESAME: an intuitive dependability modeling environment supporting inter-component dependencies. In: Proceedings of Pacific Rim International Symposium on Dependable Computing, pp. 76–83, Seoul, Korea, (2001)
44. XFDU development site. (http://sindbad.gsfc.nasa.gov/xfdu)
45. Zeginis, D., Tzitzikas, Y., Christophides, V.: On the foundations of computing deltas between rdf models. In: Proceedings of the 6th International Semantic Web Conference, ISWC/ASWC'07, pp. 637–651, Busan, Korea, November (2007)